CSE574 Project 4: Tom and Jerry in Reinforcement Learning (BONUS INCLUDED)

Nikhil Srihari
UB Person ID : **50291966**
Department of Computer Science,
SUNY – UB, Buffalo, NY
nikhilsr@buffalo.edu

Abstract

In this report, we deal with Reinforcement Learning and specifically with, Reinforcement Learning applied to Deeps Nets.

Introduction

This project was about Reinforcement Learning and Reinforcement Learning applied to Deep Nets, called Deep Q Net. We learnt the basic concepts of Reinforcement learning. As part of this project we wrote snippets of code, in from-the-scratch implementation of Deep Q Nets.We also learnt key concepts such as Q Value, Exploration, Exploitation, Epsilon decay, etc. We also learnt how to draw the State-Reward table.

Coding Tasks:

1.1.4 Coding Tasks and report

- **What parts have you implemented? What is their role in training the agent?**
  There are 3 snippets of code that we have written for this project:

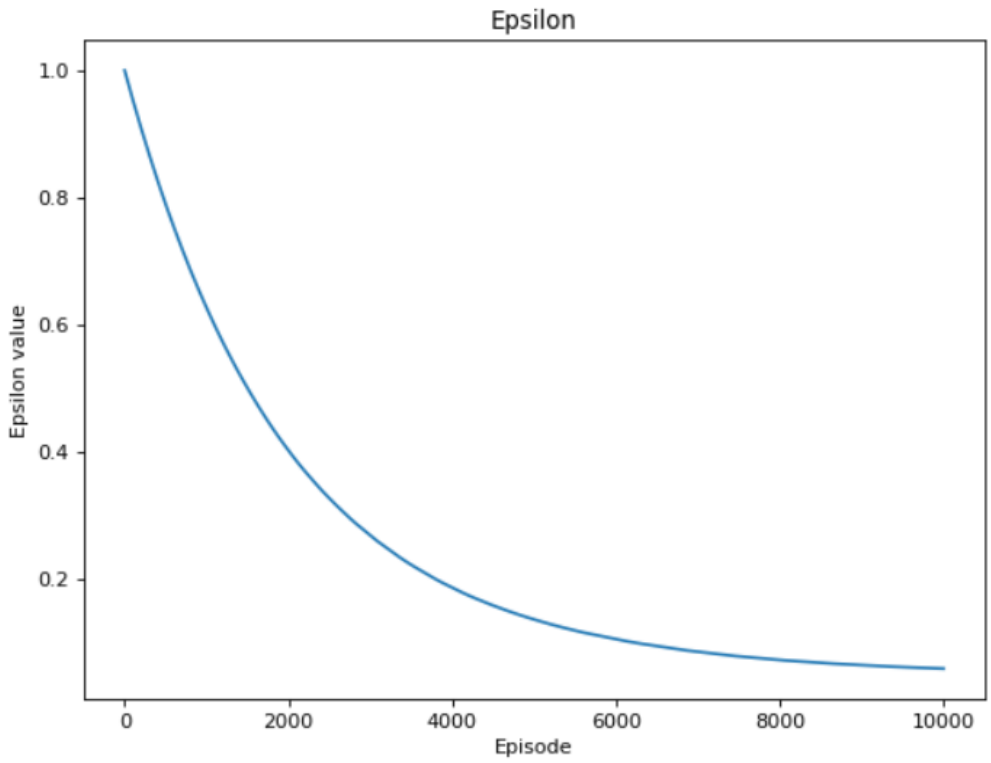
```
model.add(Dense(128, input_dim=self.state_dim))
model.add(Activation('relu'))

model.add(Dense(128))
model.add(Activation('relu'))

model.add(Dense(self.action_dim))
model.add(Activation('linear'))
```

In this snippet, we create a 3-layered Neural Network. The purpose of this project is to build a Deep Net that

learns through Reinforcement Learning, also called Deep Q Net. In this snippet we add 3 layers – 2 hidden, each with 128 nodes and 1 output layer. The activation functions used are relu in both hidden layers and then linear for the output layer. All of these layers are dense layers with no dropout – meaning that all the neurons in the previous layer connect to all the nodes in the current layer.
This Deep Q Net forms the brain of the Agent. And thus, this is responsible for the decision-making process to determine actions.

```
### START CODE HERE ### (≈ 1 line of code)
self.epsilon = self.min_epsilon + ((self.max_epsilon - self.min_epsilon)*math.exp(-1*self.lamb*abs(self.steps)))
### END CODE HERE ###
```

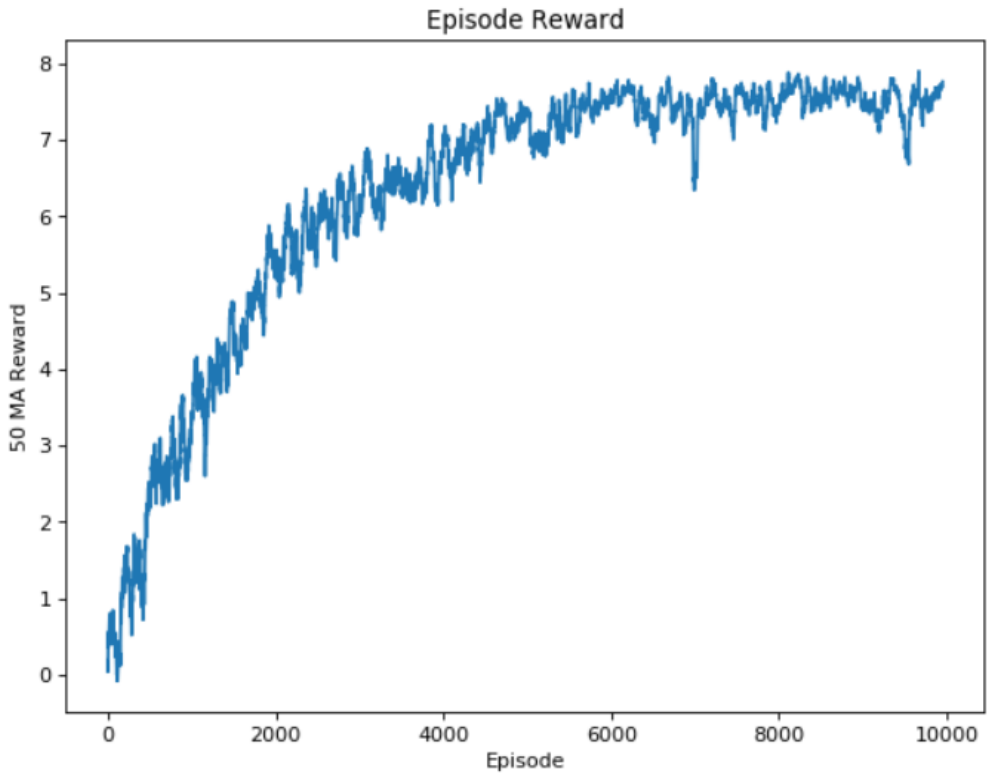
In this snippet, we code for the decay of the epsilon value. Epsilon value relates the concept of Exploration and Exploitation. During the very initial phases of our training we want our Deep Net to explore all the different possible actions. This is called Exploration. However, as the training continues, we want the Deep Net to get more and more narrower in its choice of actions – meaning we want our Deep Net to choose one action and go deeper and deeper into that action and its consequences. This is called Exploitation. Epsilon value decides whether we are Exploring or Exploiting. When Epsilon is high, the action chosen is highly randomized, i.e., we are Exploring. Thus, as time progresses, we need to reduce Epsilon value as Exploitation becomes more important than Exploration.



This is the graph showing Epsilon value decay, as resulted by our code.

```
### START CODE HERE ### (≈ 4 line of code)
if(st_next is None):
  t[act] = rew
else:
  t[act] = rew + (self.gamma) * max(q_vals_next[i])

### END CODE HERE ###
```

In this code snippet, we calculate the Q value. If there are no further steps found, i.e., we have reached termination, Qval is just the 't'th step reward value. However, if there are further steps to be evaluated, then, Q value is the sum of 2 quantities. One of those is the 't'th step reward. The second is the product of gamma and max future Q values. Given the current step, for all possible future actions we can take, we calculate the reward. And we take the max of these values, as we want to maximize reward. This is the second quantity. The future Q values ensure that the reward goes on decreasing as the number of steps keeps increasing. Below, is the graph plotting the number of episodes vs the reward:



- **Can these snippets be improved and how it will influence the training the agent?**
  We could have made 2 minor changes to better the training of the agent. Our epsilon value currently has a very steady decay. We could have modified this steady decay to add sudden spikes in the Epsilon value after a certain number of steps. For example, suddenly increase value of Epsilon by 25% after 100 steps. And then let is decay steadily as usual. Again spike it up by some 33% after 80 steps. And then let it decay again. By doing this, we let exploration happen in the later stages of learning as well.
  In our code implementation, we look only one step into the future to calculate the Q value. We could have modified this to take into account all future possible steps till termination. This would, thus, give a better reward value at step t.
- **How quickly your agents were able to learn?**
  Our agents finished learning in 10000 episodes.

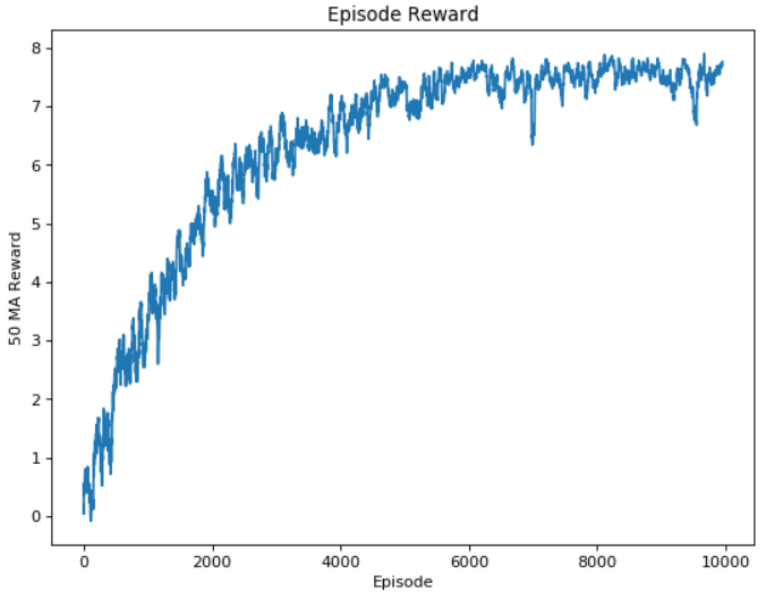
  From the graph, we can see that there is an exponential increase in reward till 4000 episodes. There is further increase, but not as quick, in reward value from 4000 to 6000 episodes. After this point, the reward value starts sort of oscillating.
  Apart from the graph, these results can also be observed by

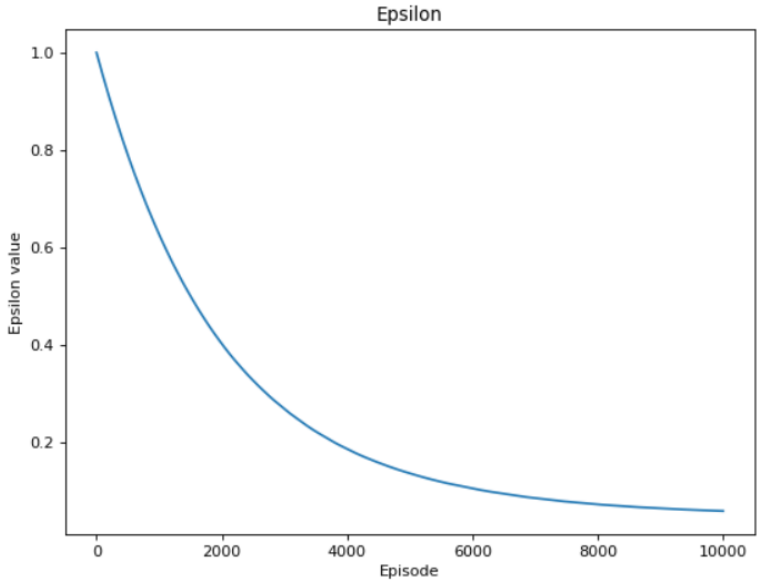
the stats printed by the program(snapshots below).

```
----------
Episode 4000
Time Elapsed: 96.27s
Epsilon 0.18679206301301599
Last Episode Reward: 7
Episode Reward Rolling Mean: 4.558318379902589
----------
Episode 6000
Time Elapsed: 141.19s
Epsilon 0.1058691296149514
Last Episode Reward: 8
Episode Reward Rolling Mean: 5.422979156075241
==========
Episode 9900
Time Elapsed: 226.78s
Epsilon 0.06028731913284897
Last Episode Reward: 8
Episode Reward Rolling Mean: 6.246607489031732
----------
```

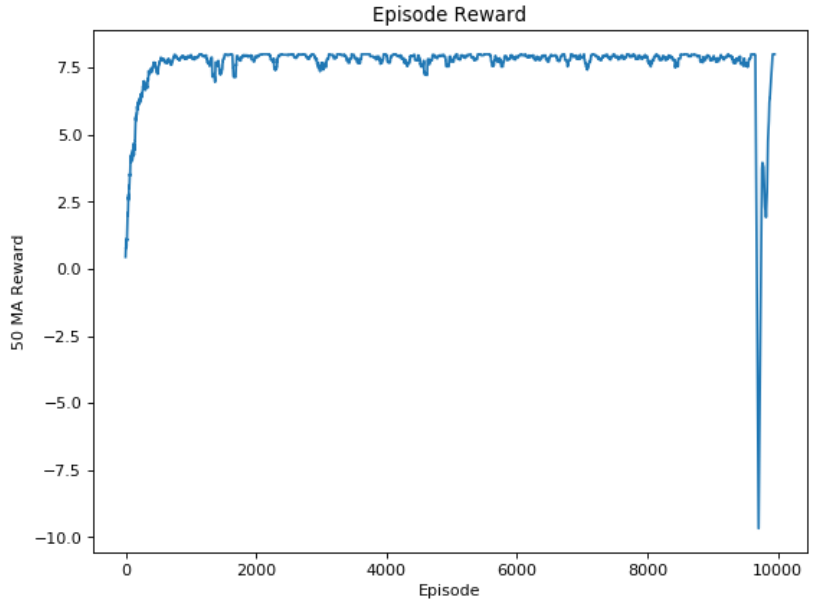
- Graphs for different values of parameters - e epsilon max/min, number of episodes, gamma etc

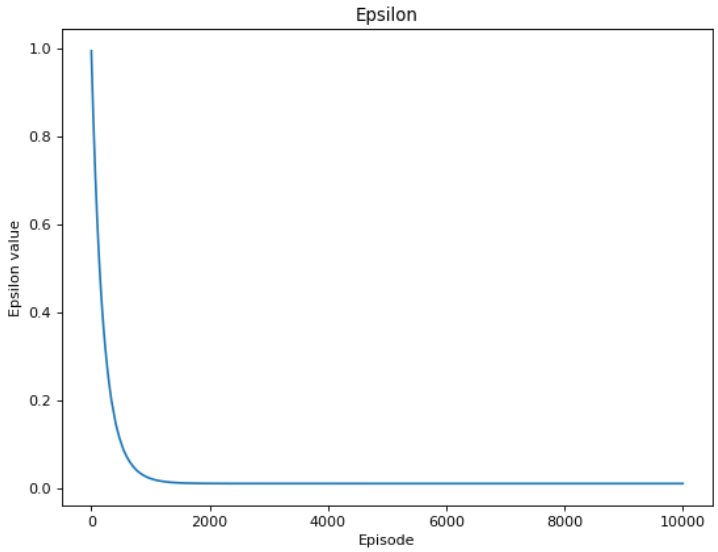
1. MAX_EPSILON = 1; MIN_EPSILON = 0.05; LAMBDA = 0.00005; num_episodes = 10000; gamma = 0.99



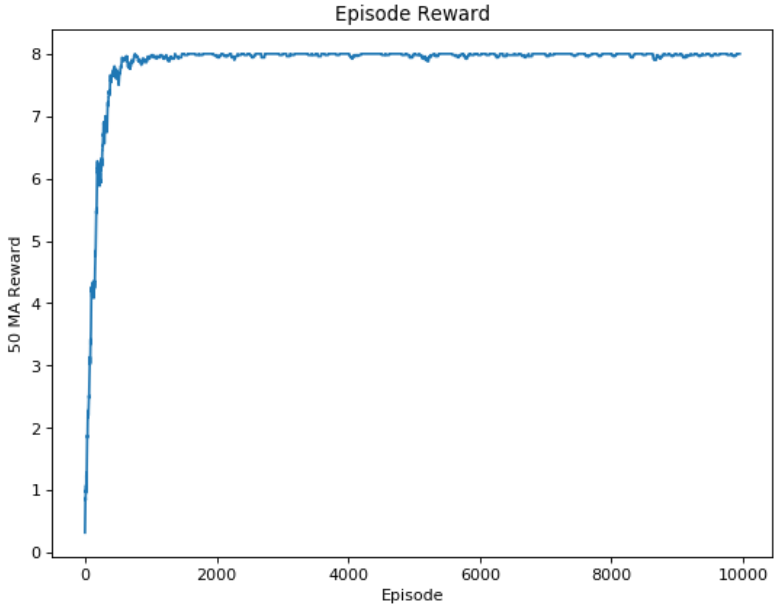


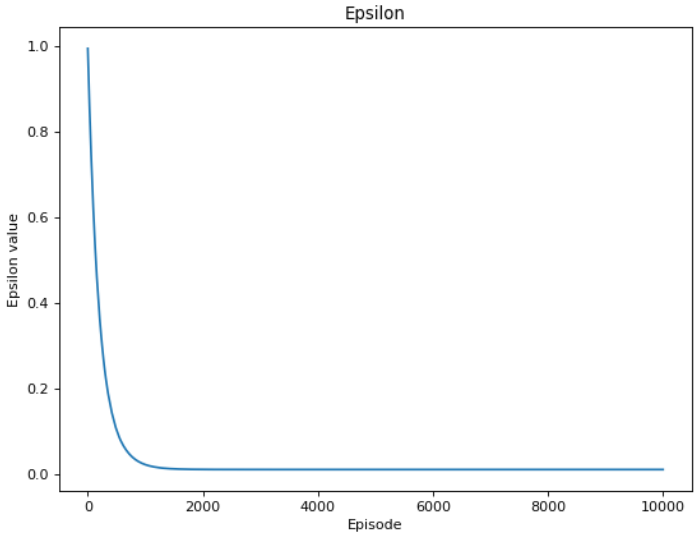
2. MAX_EPSILON = 1;MIN_EPSILON = 0.01;LAMBDA = 0.0005; num_episodes = 10000; gamma = 0.99





3. MAX_EPSILON = 1;MIN_EPSILON = 0.01;LAMBDA = 0.0005; num_episodes = 10000; gamma = 0.6





Writing tasks:

1. **Explain what happens in reinforcement learning if the agent always chooses the action that maximizes the Q-value. Suggest two ways to force the agent to explore.**

   Q value gives us the reward for every action at a given state and let's us choose the best action which maximizes reward. While the value function tells us how good we estimate each action to be, the policy is the function that determines which actions we end up taking. Intuitively, we might want to use a policy that picks the action with the highest Q value. This performs poorly in practice, as our Q estimates will be very wrong at the start before we gather enough experience through trial and error. We need to explore all the different paths/actions before we deep dive into one set of actions. This is why we need to add a mechanism to our policy to encourage exploration.

   Two ways to force the agent to explore are:

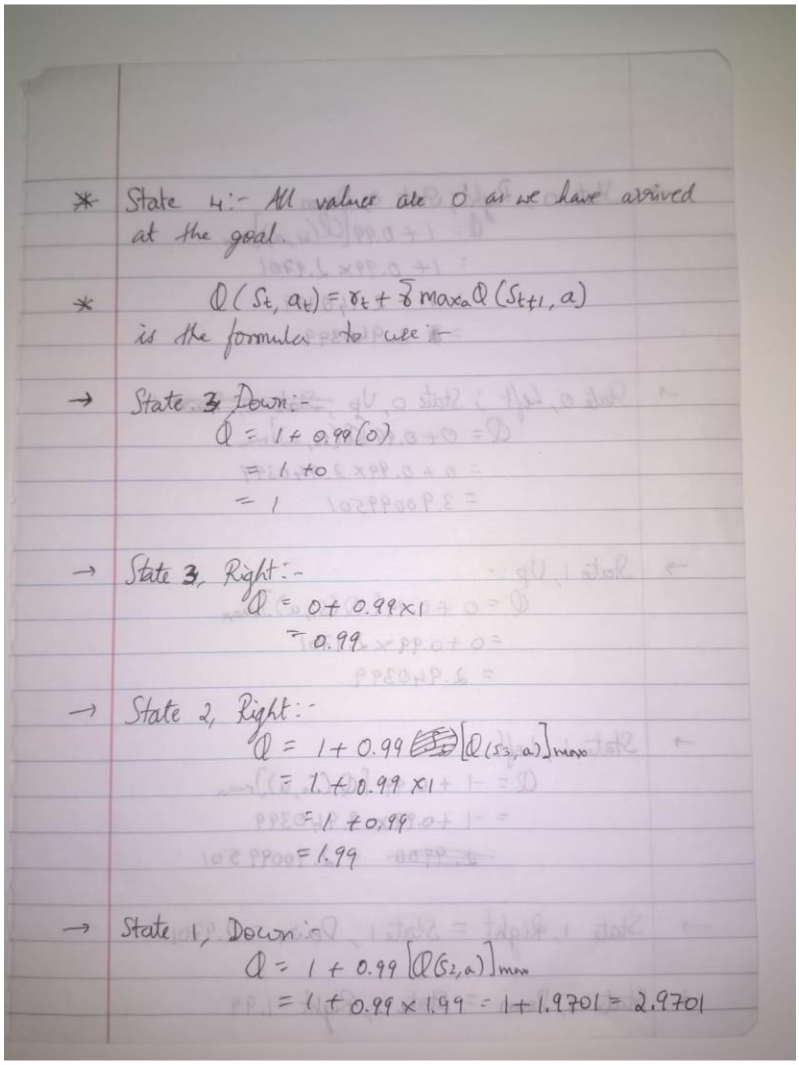
   i. We can use epsilon greedy, which consists of taking a random action with probability epsilon. We start with epsilon being close to 1, always choosing random actions, and lower epsilon as we go along and learn more about which chests are good. Eventually, we learn which chests are best.

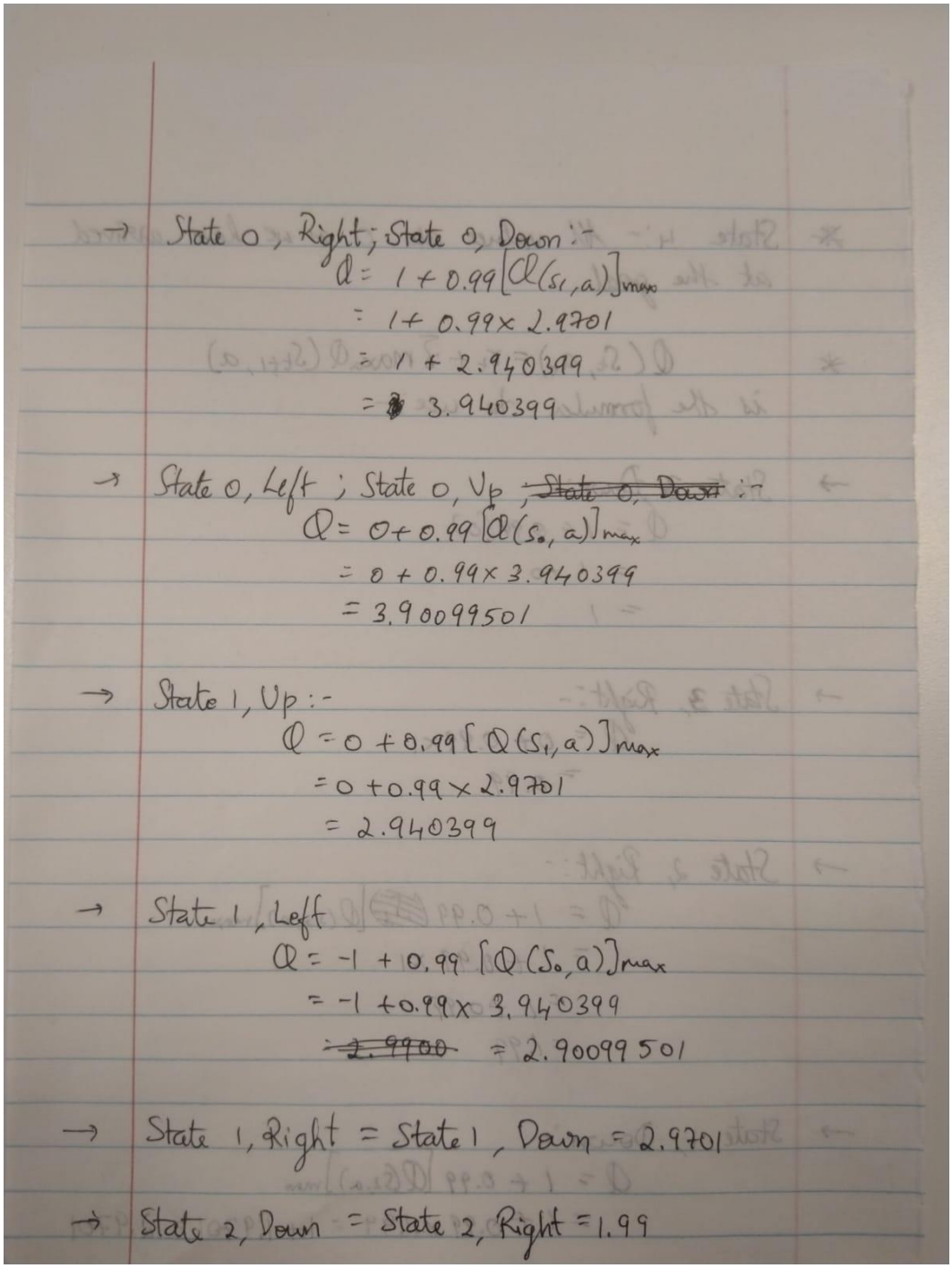
   ii. In practice, however, Boltzmann Exploration is a popular method, which adjust probabilities based on our current estimate of how good each chest is, adding in a randomness factor. This is a more subtle approach than either taking the action we think is the best, or a random action.

2. **Calculate Q-value for the given states and provide all the**

calculation steps.

STATE	UP	DOWN	LEFT	RIGHT
0	3.900995	3.940399	3.900995	3.94039
1	2.940399	2.9701	2.900995	2.9701
2	1.940399	1.99	1.940399	1.99
3	0.9701	1	0.9701	0.99
4	0	0	0	0





→ State 0, Right; State 0, Down :-

$$Q = 1 + 0.99[Q(s_1, a)]_{max}$$
$$= 1 + 0.99 \times 2.9701$$
$$= 1 + 2.940399$$
$$= 3.940399$$

→ State 0, Left ; State 0, Up ~~; State 0, Down~~ :-

$$Q = 0 + 0.99[Q(s_0, a)]_{max}$$
$$= 0 + 0.99 \times 3.940399$$
$$= 3.90099501$$

→ State 1, Up :-

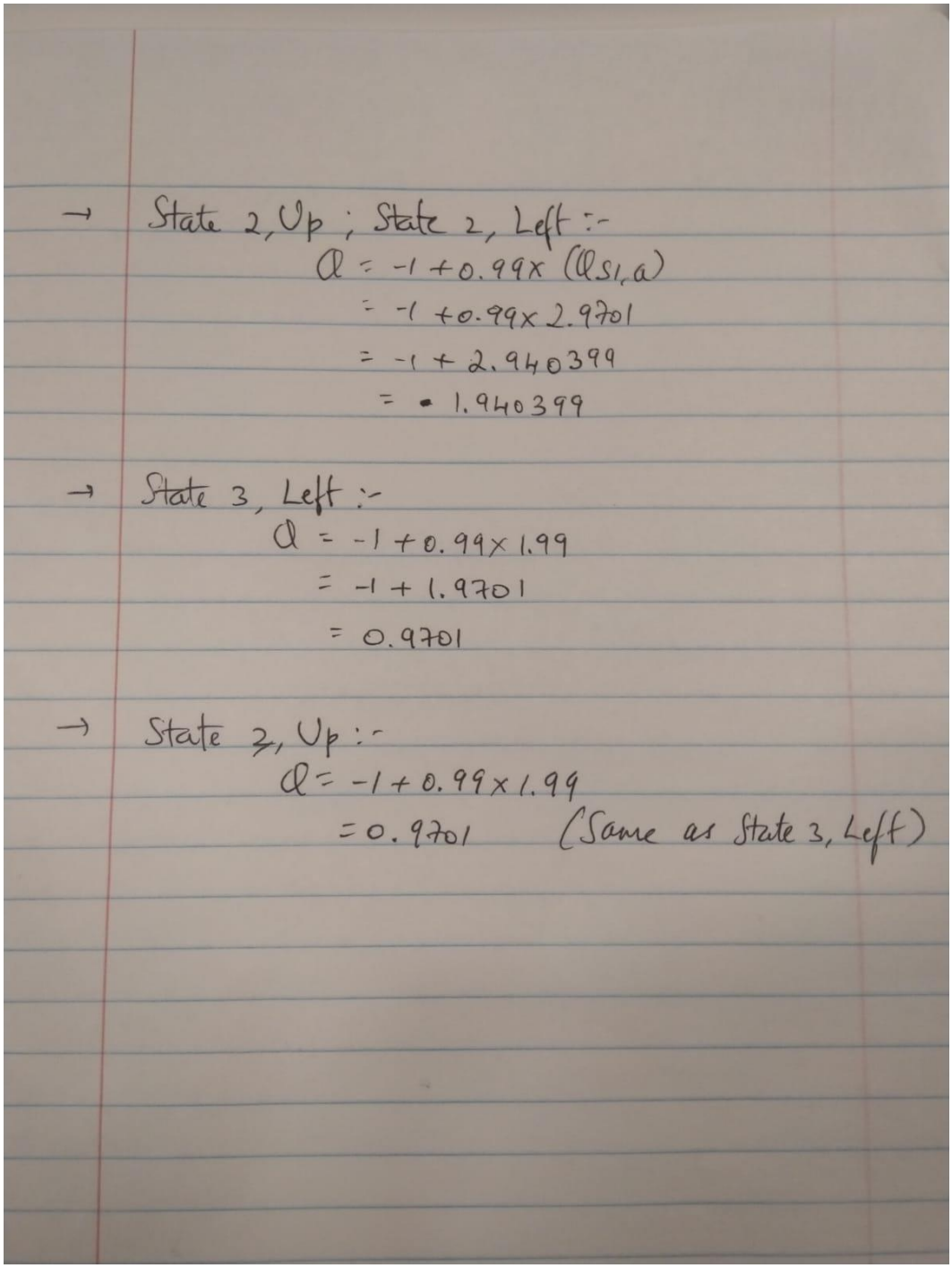
$$Q = 0 + 0.99[Q(s_1, a)]_{max}$$
$$= 0 + 0.99 \times 2.9701$$
$$= 2.940399$$

→ State 1, Left

$$Q = -1 + 0.99[Q(s_0, a)]_{max}$$
$$= -1 + 0.99 \times 3.940399$$
$$\text{~~2.9900~~} = 2.90099501$$

→ State 1, Right = State 1, Down = 2.9701

→ State 2, Down = State 2, Right = 1.99



BONUS:

I have written the bonus code but wasn't able to run the code. The code is correct, as verified by Stable Baselines site.

References:

1. https://blog.insightdatascience.com/reinforcement-learning-from-scratch-819b65f074d8
2. https://medium.com/@m.alzantot/deep-reinforcement-learning-demysitifed-episode-2-policy-iteration-value-iteration-and-q-978f9e89ddaa